22074661-25533

Application

For

United States Letters Patent

To all whom it may concern:

Be it known that we,

Tad Deffler, and

Eric Mintz

have invented certain new and useful improvements in

METHOD AND SYSTEM FOR AN EXTENSIBLE MACRO LANGUAGE

of which the following is a full, clear and exact description:

Eunhee Park
Reg. No. 42,976
Baker & McKenzie
805 Third Avenue
New York, NY 10022

09417249 . 101599

[Docket No. 22074661-25533]

METHOD AND SYSTEM FOR AN EXTENSIBLE MACRO LANGUAGE

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application claims benefit of the filing date of U.S. Patent Application No. 60/104,682 entitled MODELING TOOL SYSTEMS AND METHODS, filed on October 16, 1998.

The present application is related to co-pending U.S. Patent Application No. 09/419,736 ~~(Attorney Docket #22074661-25531)~~ entitled METHOD FOR DETERMINING DIFFERENCES BETWEEN TWO OR MORE MODELS, being concurrently filed on the same day, which is incorporated by reference herein in its entirety.

The present application is related to a co-pending U.S. Patent Application No. 09/419,731 ~~(Attorney Docket #22074661-25532)~~ entitled METHOD FOR IMPACT ANALYSIS OF A MODEL, being concurrently filed on the same day, which is incorporated by reference herein in its entirety.

The present application is related to co-pending U.S. Patent Application No. 09/418,751 ~~(Attorney Docket #22074661-25534)~~ entitled METHOD AND APPARATUS FOR PROVIDING ACCESS TO A HIERARCHICAL DATA STORE THROUGH AN SQL INPUT, being concurrently filed on the same day, which is incorporated by reference herein in its entirety.

a

a

The present application is related to a co-pending U.S. Patent Application No. 09/420,223 ~~(Atty. Docket #22074661-25535)~~ entitled APPARATUS AND METHOD FOR MODELING TOOLS, being concurrently filed on the same day, which is incorporated by reference herein in its entirety.

DESCRIPTION

TECHNICAL FIELD OF THE INVENTION

The present invention relates in general to computer language processors and, particularly to an extensible macro language.

BACKGROUND OF THE INVENTION

A macro is a set of commands that can be played back to perform a given task. Examples of these tasks include inserting a commonly used name and address into a word processor or executing a series of keystrokes to format a file. Tasks performed by macros are typically repetitive in nature allowing significant savings in time by executing the macro instead of manually repeating the commands.

Currently, different applications allow users to write macros and scripts within the confines of the allowed domain, i.e., within the confines of the specific application. For example, word processors typically allow users to create a macro by recording series of keystrokes to be played back later. Other applications allow users to create macros for retrieving and manipulating data within the confines of the applications. Thus, these applications include a limited set of macros, e.g., macro for recording keystrokes, a macro for retrieving data. The user is then typically limited to the macros provided by the application.

Frequently, however, each user using an application has a unique set of instructions or commands that the user would like to include as a macro in the application which was not previously provided. Because the macros are typically hard coded into the applications or into the macro language included in the applications, the only available method currently available to include additional macros into the application is to hard code the new macros into the application by modifying the source code and recompiling it before the new macro can be used. Usually, however, this presents a problem because the user is not given an access to the source code of the macro language or the application to modify. Moreover, it would be a tremendous burden on the application developers to try to cater to each individual user's preferences by customizing the applications to include the macros that the user would like to have.

Therefore, it is highly desirable to have an extensible macro language that would allow users to modify and extend the language to include their preferences when using the macro language. Furthermore, it is also highly desirable to be able to allow the users to extend the macro without having to modify or access the source code of the macro language since the source code is treated as a proprietary property not distributed to the users.

SUMMARY OF THE INVENTION

To overcome the above shortcomings of the prior art macro language processors the present invention provides an extensible macro language that allows users to write new macro commands that include procedures tailored to the specific needs of the users without a need to modify any source code of the macro language processor. The extensible macro language is enabled to process the new macro commands by recognizing the new macro commands unknown to the language and associating the new macro commands with procedure calls stored in a registry, i.e., a repository, thereby allowing dynamic extension of a macro language.

In the present invention, a mechanism for dynamically registering new macro commands in a registry is also provided for allowing extensibility. To register new macro commands, the users may insert keywords representing the new macro commands and the associated codes or procedures in the registry for execution by the extensible macro language.

The present invention also defines a simplistic syntax for the extended macro language for recognizing the new macro commands for what they are without needing to know what functions they perform.

According to the goals of the present invention, there is provided a parser and a macro handler for processing macro commands not previously defined in the macro language. The macro commands not previously defined or undefined in the macro language refer to those macro commands that were not included in the set of commands available in the macro language at the time of release and distribution to the users. The parser analyzes keywords in a macro language expression and recognizes one or more keywords representing macro commands that were not previously defined in the macro language. The macro handler receives the keyword in the macro expression and retrieves from a registry of keywords, an executable code associated with the keyword. The executable code is run to process the macro command represented by the keyword. The registry of keywords may be augmented to include any keywords and associated codes for extending the macro language.

Further features and advantages of the present invention as well as the structure and operation of various embodiments of the present invention are described in detail below with reference to the accompanying drawings.

In the drawings, like reference numbers indicate identical or functionally similar elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 is a block diagram illustrating the components of the extensible macro language of the present invention; and

Figure 2 illustrates an example of a macro expression having an iterator macro.

DETAILED DESCRIPTION OF THE INVENTION

The present invention is directed to an extensible macro language which may be extended dynamically in the runtime environment without having to rebuild and recompile the macro language. Although the extensible macro language may include a predetermined set of macro commands, the present invention allows users to add additional or new macro commands as desired. Figure 1 is a block diagram 100 illustrating the components of the system for providing the extensible macro language of the present invention. The parser 102 includes a grammar or syntax 104 that the parser 102 employs to analyze and parse a given expression. As shown in Figure 1, the parser 102 receives a macro language expression 106 and parses the expression into components according to the syntax 104 of the macro language. The syntax used in one embodiment of the present invention will be described in more detail hereinbelow. Referring back to Figure 1, the parser 102 reads the expression 106 recognizing certain tokens predefined in the syntax that indicate a presence of a new macro command. In this example, when the parser 102 encounters curly braces in the expression 106, parser 102 treats the keywords, for example, "property (name)", embedded within the braces as a new macro command. Moreover, the parser 102 recognizes, based on the syntax 104, that the "name" embedded within the parenthesis is a parameter to the new macro command. Other aspects of the syntax 104 may dictate that a string of characters outside any symbols to be interpreted as a literal string. Accordingly, the parser 102 breaks each element in the expression into components as shown at 108.

A novel feature of the parser 102 in the present invention is that the parser 102 is transparent to the actual content within the tokens, i.e., curly braces. That is, as long as the new macro commands or keywords are embedded within a recognizable token, the parser 102 breaks the keywords down into components regardless of whether the keywords have been predefined in the macro language. Thus, as shown at 108, the macro expression 106 is broken down into components according to the syntax 104 of the extended macro language. The new keyword "property" is broken down as a token component 108a; the string "name" within the parenthesis is broken down as a parameter component 108b; the string "likes" is broken down as a literal component 108c; and the string "pizza" is also broken down as a literal component 108d.

As shown in Figure 1, the present invention also includes a macro handler 110, and a repository 112 having keywords and their corresponding executable codes. The executable codes may be stored in the repository 112 as a pointer to the actual codes 114 for execution. The repository 112 includes one or more keywords and associated codes, and may be dynamically modified, e.g., new keywords and codes added to it as need arises by a user of the macro language. The repository 112 in the present invention may be a simple file with a table of keywords and associated codes. Alternatively, a separate database may be used as the repository 112.

After the macro expression has been parsed into separate components as described above with reference to the parser 102, the components are then passed to the macro handler 110 for additional processing. For the token component having the keyword "property" 108a, the macro handler checks a repository to the keyword "property". If found, the code associated with the keyword "property" is retrieved and executed. In executing the code, the macro handler 110 passes all the parameters found in the macro expression and parsed as parameters, to the executing code. The macro handler 110 does not need to know any other processing that may be performed inside the code itself. All that the macro handler 110 needs to recognize is that the "property" is a keyword to be looked up in the repository 112 for its corresponding code, and the specified corresponding code in the repository 112 to be executed with any parameters. The corresponding code is typically specified in the repository 112 as a pointer to the actual code itself 114.

After the proper execution of the code 114 specified in the repository, the macro handler 110 accepts one or more outputs, if any, of the executed code and places the outputs back into the macro expression in place of the keyword. Thus, in the example shown in Figure 1, the output of the code associated with the "property" with the parameter "name" may be MARY. Consequently, the result of the extended macro expression "{property (name)} likes pizza" at 106 is "Mary likes pizza" as shown at 116.

A novel feature of the present invention is that the macro handler, like the parser, need not know anything in the code or what type of functions are being performed by the executable code. The macro handler merely provides an initiation into the executable code that is associated with the keyword. In an exemplary embodiment of the present invention, it is left up to the users to define exactly what the code should do, and consequently, therefore, what command the keyword is to perform, thus providing a flexible and extensible macro language.

In the above example, the output MARY may have been obtained in various ways transparent to the macro language. For example, the name MARY may have been obtained by performing a search from the World Wide Web, or may have been obtained from a database using a query language, further illustrating the extensibility afforded by the present invention.

<u>The Language Syntax</u>

The syntax or the grammar employed in one embodiment of the extensible macro language will now be described in detail. The extensible macro language of the present invention includes a syntax (Figure 1 104) comprising literals, macros, comments and operator/scoping characters.

<u>Literal</u>

The syntax in this embodiment treats all text outside of curly braces as a literal, and is emitted exactly as typed. Within curly braces, text inside double quotes is treated as a literal. Such a scheme allows for embedding of a literal within macro calls. Some examples of a literal are illustrated as follows:

```
This text would be emitted just like this;
```

{"So would this text"}

Macros

Macros include instructions to the macro processor, like procedures or functions in a programming language. According to the syntax defined in the present invention, all macros are embedded within curly braces. In one embodiment, the macros fall into two categories: procedures macros and iterator macros.

Procedure macros are designed to perform some work. They may expand to a value; they may declare a variable; they may invoke a process. The actions performed are entirely specified by the designer of the macro. In one embodiment, the macros must, however, return a "true" value upon successful completion of their task and a "false" value upon failure.

The following expression illustrates a string literal, followed by a macro call for getting the page number when printing:

```
My Model Report - Page {HeaderPage}      : Input
My Model Report - Page 1                 : Output
```

In the above example, the HeaderPage is a macro defined by a user to extract a page number.

Iterator macros allow the user to traverse across data structures. Iterators are distinguished by the keywords "begin" and "end" that delimit a block of code following the iterator declaration. The code within the "begin/end" block is executed once for each iteration. When the iterator has moved across all objects in its pool, control breaks out of the iteration block and continues to execute a next statement in the macro expression after the block.

The following block of macro expression illustrates a use of the iterator macro:

```
{
        MyIterator
        begin
                DoSomething
        end
}
```

In the above example, the procedure macro "DoSomething" executes once for each element returned by the "MyIterator" macro. The curly braces surrounding the entire fragment indicates that all expression within the braces is to be treated as macro code.

<u>Parameters</u>

The syntax defined in the extensible macro language of the present invention allows for both procedure and iterator to accept and process parameters. Parameters may include strings, or other macros. To distinguish parameters, the parameters are enclosed within parenthesis following the macro. Macros may accept variable-length parameter lists, as desired. The following illustrates a macro expression having a parameter "foo":

```
{MacroWithParameters ("foo")}
```

Control blocks

In some instances, it is desirable to have a block of a macro expression to fail if any portion of it fails. The following example illustrates one such instance:

```
{FirstName [MiddleInitial "."] LastName}
```

If there was no middle initial, the MiddleInitial macro would return a nil value or a fail value. In that case, the literal "." should not be printed. To accommodate for such conditions, the present invention includes in its syntax, square brackets ("[]") that denote a conditional expression. Thus, if the macro within the square brackets fails, the rest of the expression in the square brackets is not emitted. In the above example, if the MiddleInitial failed for lack of value, the literal "." is not be printed.

The conditional blocks have internal scope, i.e., the failure of a conditional block does not affect the surrounding code. For conditions in a block to affect the outer block, the syntax additionally includes what is referred to as a propagating conditional denoted by angle brackets. If any macro within a pair of angle brackets fails, the block within the angle brackets as well as the next outer block fails. The following examples illustrate macro expression with a conditional and a propagating conditional:

```
{ Print " " [ Print [ Fail ] ] }      : input
foo foo                                 : output

{ Print " " [ Print < Fail > ] }      : input
foo                                     : output
```

In both examples the "Print" macro outputs the word "foo". In the first example, the failed macro in square brackets is contained within its block. Thus, the next outer block having "Print" is executed as well as the first "Print", resulting in the output "foo foo". In the second example, when a macro within angle brackets fails, the failure is propagated to the next block having the "Print" macro. Thus, the next outer block with "Print" is not executed. Since this Print macro is contained within a pair of square brackets, the failure is contained in the block. Thus, the first "Print" macro is executed, resulting in the output "foo".

Figure 2 illustrates an example of a macro expression including an iterator macro of the present invention. As described with reference to Figure 1, the keyword "ForEach" is recognized by the parser 102 (Figure 1) as a macro, and the word "Employee" is recognized as a parameter to the macro "ForEach". When the macro handler receives the token keyword "ForEach", the macro handler 110 (Figure 1) performs a look-up of the keyword "ForEach" in the registry 112 and executes the corresponding code. The code for "ForEach" macro, for example, may include instructions to perform commands found within the begin/end block of the macro expression for all sub-objects 204b, 204c in a given object 204 having the type of the specified parameter "employee". In this macro expression 202, another macro exists within the begin/end block. Accordingly, the macro handler 110 (Figure 1) performs a look-up of the keyword "Property" in the registry 112 and executes the corresponding code for each of the sub-objects 204b, 204c having employee type as specified in the "ForEach" keyword. The code associated with the "Property" keyword, for example, may include instructions to print the value of the type specified in the parameter of the keyword "Property", in this case, an employee name as specified by "EmpName". Consequently, the result of the macro expression 202 is the output shown at 208, "Mary John".

The extensible macro language of the present invention is useful for customizing macros specific to the needs of individual users. For example, the extensible macro language has been interfaced with the UMA Model for retrieving various objects from the UMA Model, as desired by a user. The UMA is disclosed in a co-pending U.S. Patent Application No. 09/420,223 ~~(Atty. Docket #22074661-25535)~~ entitled APPARATUS AND METHOD FOR MODELING TOOLS, filed on October 15, 1999, the disclosure of which is incorporated herein by reference in its entirety thereto. Appendix A includes a brief description of the extensible macro language of the present invention as used in the UMA Model and referred to as the UMA Template Language. The description in Appendix A explains one embodiment of the extensible macro language and should in no way be read as limiting the scope and capabilities of the extensible macro language to the descriptions contained therein.

While the invention has been particularly shown and described with respect to an embodiment thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and details may be made therein without departing from the spirit and scope of the invention.